

THE F!#%ING MANUAL

SCRIBUS API COMMANDS

Page 3 : documents related commands
Page 7 : page related commands
Page 11 : creating and destroying objects
Page 15 : selecting objects
Page 17 : setting objects proprieties
Page 21 : getting objects proprieties
Page 23 : manipulating objects
Page 25 : handling text frames
Page 29 : color related commands
Page 31 : font related commands
Page 33 : layer related commands
Page 37 : using dialogus from scribus

Usage examples:

```
result = MessageBox [49]('Script failed',  
    'This script only works when you have a text frame selected.',  
    ICON_ERROR)  
  
result = MessageBox [50]('Monkeys!', 'Something went ook! <i>Was it a monkey?</i>',  
    ICON_WARNING, BUTTON_YES|BUTTONOPT_DEFAULT,  
    BUTTON_NO, BUTTON_IGNORE|BUTTONOPT_ESCAPE)
```

Defined button and icon constants: BUTTON_NONE, BUTTON_ABORT, BUTTON_CANCEL, BUTTON_IGNORE, BUTTON_NO, BUTTON_NOALL, BUTTON_OK, BUTTON_RETRY, BUTTON_YES, BUTTON_YESALL, ICON_NONE, ICON_INFORMATION, ICON_WARNING, ICON_CRITICAL.

newDocDialog(...)

newDocDialog() -> boolDisplays the "New Document" dialog box. Creates a new document if the user accepts the settings. Does not create a document if the user presses cancel. Returns true if a new document was created.

newStyleDialog(...)

newStyleDialog() -> stringShows 'Create new paragraph style' dialog. Function returns real style name or None when user cancels the dialog.

statusMessage(...)

messageBarText("string")Writes the "string" into the Scribus message bar (status line). The text must be UTF8 encoded or 'unicode' string(recommended).

progressReset(...)

progressReset()Cleans up the Scribus progress bar previous settings. It is called before the new progress bar use. See progressSet.

progressSet(...)

progressSet(nr)Set the progress bar position to "nr", a value relative to the previously set progressTotal. The progress bar uses the concept of steps; you give it the total number of steps and the number of steps completed so far and it will display the percentage of steps that have been completed. You can specify the total number of steps with progressTotal [51](). The current number of steps is set with progressSet [52](). The progress bar can be rewound to the beginning with progressReset [53](). [based on info taken from Trolltech's Qt docs]

progressTotal(...)

progressTotal(max)Sets the progress bar's maximum steps value to the specified number. See progressSet.

valueDialog(...)

valueDialog(caption, message [defaultvalue]) -> stringShows the common 'Ask for string' dialog and returns its value as a string Parameters: window title, text in the window and optional 'default' value.

Example: valueDialog [54]('title', 'text in the window', 'optional')

Document related Commands

closeDoc(...)

closeDoc()Closes the current document without prompting to save. May throw NoDocOpenError [47] if there is no document to close

docChanged(...)

docChanged(bool)Enable/disable save icon in the Scribus icon bar and the Save menu item. It's useful to call this procedure when you're changing the document, because Scribus won't automatically notice when you change the document using a script.

getUnit(...)

Unit() -> integer (Scribus unit constant)Returns the measurement units of the document. The returned value will be one of the UNIT_* constants: UNIT_INCHES, UNIT_MILLIMETERS, UNIT_PICAS, UNIT_POINTS.

haveDoc(...)

haveDoc() -> boolReturns true if there is a document open.

loadStylesFromFile(...)

loadStylesFromFile("filename")Loads paragraph styles from the Scribus document at "filename" into the current document.

masterPageNames(...)

masterPageNames()Returns a list of the names of all master pages in the document.

newDoc(...)

newDoc(size, margins, orientation, firstPageNumber, unit, facingPages, firstSideLeft) -> boolWARNING: Obsolete procedure! Use newDocument [48] instead. Creates a new document and returns true if successful. The parameters have the following meaning:

- size = A tuple (width, height) describing the size of the document. You can use predefined constants named PAPER_<paper_type> e.g. PAPER_A4 etc.
- margins = A tuple (left, right, top, bottom) describing the document margins
- orientation = the page orientation - constants PORTRAIT, LANDSCAPE
- firstPageNumber = is the number of the first page in the document used for page numbering. While you'll usually want 1, it's useful to have higher numbers if you're creating a document in several parts.
- unit: this value sets the measurement units used by the document. Use a predefined constant for this, one of: UNIT_INCHES, UNIT_MILLIMETERS, UNIT_PICAS, UNIT_POINTS.
- facingPages = FACINGPAGES, NOFACINGPAGES
- firstSideLeft = FIRSTPAGELEFT, FIRSTPAGERIGHT

The values for width, height and the margins are expressed in the given unit for the document. PAPER_* constants are expressed in points. If your document is not in points,

make sure to account for this.

example: newDoc [49](PAPER_A4, (10, 10, 20, 20), LANDSCAPE, 1, UNIT_POINTS, FACINGPAGES, FIRSTPAGERIGHT)

newDocument(...)

newDocument(size, margins, orientation, firstPageNumber, unit, pagesType, firstPageOrder, numPages) -> boolCreates a new document and returns true if successful. The parameters have the following meaning:

- size = A tuple (width, height) describing the size of the document. You can use predefined constants named PAPER_<paper_type> e.g. PAPER_A4 etc.
- margins = A tuple (left, right, top, bottom) describing the document margins
- orientation = the page orientation - constants PORTRAIT, LANDSCAPE
- firstPageNumber = is the number of the first page in the document used for pagenumbering. While you'll usually want 1, it's useful to have higher numbers if you're creating a document in several parts.
- unit: this value sets the measurement units used by the document. Use a predefined constant for this, one of: UNIT_INCHES, UNIT_MILLIMETERS, UNIT_PICAS, UNIT_POINTS.
- pagesType = One of the predefined constants PAGE_n. PAGE_1 is single page, PAGE_2 is for double sided documents, PAGE_3 is for 3 pages fold and PAGE_4 is 4-fold.
- firstPageOrder = What is position of first page in the document. Indexed from 0 (0 = first).
- numPage = Number of pages to be created.

The values for width, height and the margins are expressed in the given unit for the document. PAPER_* constants are expressed in points. If your document is not in points, make sure to account for this.

example: newDocument(PAPER_A4, (10, 10, 20, 20), LANDSCAPE, 7, UNIT_POINTS, PAGE_4, 3, 1)

May raise ScribusError if is firstPageOrder bigger than allowed by pagesType.

openDoc(...)

openDoc("name")Opens the document "name".

May raise ScribusError if the document could not be opened.

placeEPS(...)

placeEPS("filename", x, y) Places the EPS "filename" onto the current page, x and y specify the coordinate of the topleft corner of the EPS placed on the page If loading was successful, the selection contains the imported EPS

placeODG(...)

placeODG("filename", x, y) Places the ODG "filename" onto the current page, x and y specify the coordinate of the topleft corner of the ODG placed on the page If loading was successful, the selection contains the imported ODG

placeSVG(...)

placeSVG("filename", x, y) Places the SVG "filename" onto the current page, x and y specify the coordinate of the topleft corner of the SVG placed on the page If loading was successful, the selection contains the imported SVG

Using Dialogs from Scribus

fileDialog(...)

fileDialog("caption", ["filter", "defaultname", haspreview, issave, isdir]) -> string with filenameShows a File Open dialog box with the caption "caption". Files are filtered with the filter string "filter". A default filename or file path can also be supplied, leave this string empty when you don't want to use it. A value of True for haspreview enables a small preview widget in the FileSelect box. When the issave parameter is set to True the dialog acts like a "Save As" dialog otherwise it acts like a "File Open Dialog". When the isdir parameter is True the dialog shows and returns only directories. The default for all of the optional parameters is False.

The filter, if specified, takes the form 'comment (*.type *.type2 ...)'. For example 'Images (*.png *.xpm *.jpg)'.

Refer to the Qt-Documentation for QFileDialog for details on filters.

Example: fileDialog [47]('Open input', 'CSV files (*.csv)')

Example: fileDialog [48]('Save report', defaultname='report.txt', issave=True)

fileQuit(...)

fileQuit()Quit Scribus.

getGuiLanguage(...)

getGuiLanguage() -> stringReturns a string with the -lang value.

messagebarText(...)

messagebarText("string")Writes the "string" into the Scribus message bar (status line). The text must be UTF8 encoded or 'unicode' string(recommended).

messageBox(...)

messageBox("caption", "message", icon=ICON_NONE,

button1=BUTTON_OK|BUTTONOPT_DEFAULT, button2=BUTTON_NONE,

button3=BUTTON_NONE) -> integerDisplays a message box with the title "caption", the

message "message", and an icon "icon" and up to 3 buttons. By default no icon is used and a single button, OK, is displayed. Only the caption and message arguments are required, though setting an icon and appropriate button(s) is strongly recommended. The message text may contain simple HTML-like markup.

Returns the number of the button the user pressed. Button numbers start at 1.

For the icon and the button parameters there are predefined constants available with the same names as in the Qt Documentation. These are the BUTTON_* and ICON_* constants defined in the module. There are also two extra constants that can be binary-ORed with button constants:

- BUTTONOPT_DEFAULT Pressing enter presses this button.

- BUTTONOPT_ESCAPE Pressing escape presses this button.

placeSXD(...)

placeSXD("filename", x, y) Places the SXD "filename" onto the current page, x and y specify the coordinate of the topleft corner of the SXD placed on the page. If loading was successful, the selection contains the imported SXD

saveDoc(...)

saveDoc() Saves the current document with its current name, returns true if successful. If the document has not already been saved, this may bring up an interactive save file dialog. If the save fails, there is currently no way to tell.

saveDocAs(...)

saveDocAs("name") Saves the current document under the new name "name" (which may be a full or relative path).
May raise ScribusError if the save fails.

setInfo(...)

setInfo("author", "info", "description") -> bool Sets the document information. "Author", "Info", "Description" are strings.

setMargins(...)

setMargins(lr, rr, tr, br) Sets the margins of the document, Left(lr), Right(rr), Top(tr) and Bottom(br) margins are given in the measurement units of the document - see UNIT_<type> constants.

setDocType(...)

setDocType(facingPages, firstPageLeft) Sets the document type. To get facing pages set the first parameter to FACINGPAGES, to switch facingPages off use NOFACINGPAGES instead. If you want to be the first page a left side set the second parameter to FIRSTPAGELEFT, for a right page use FIRSTPAGERIGHT.

setUnit(...)

setUnit(type) Changes the measurement unit of the document. Possible values for "unit" are defined as constants UNIT_<type>.
May raise ValueError if an invalid unit is passed.

setBaseLine(...)

setBaseLine(grid, offset) Sets the base line settings of the document, grid spacing(grid), grid offset(offset). Values are given in the measurement units of the document - see UNIT_<type> constants.

scrollDocument(...)

scrollDocument(x,y) Scroll the document in main GUI window by x and y.

zoomDocument(...)

zoomDocument(double) Zoom the document in main GUI window. Actions have whole number values like 20.0, 100.0, etc. Zoom to Fit uses -100 as a marker.

name isn't acceptable.

setLayerTransparency(...)

setLayerTransparency("layer", trans) Sets the layers "layer" transparency to trans. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setLayerVisible(...)

setLayerVisible("layer", visible) Sets the layer "layer" to be visible or not. If is the visible set to false the layer is invisible. May raise NotFoundError [53] if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

isLayerPrintable(...)

isLayerPrintable("layer") -> bool>Returns whether the layer "layer" is printable or not, a value of True means that the layer "layer" can be printed, a value of False means that printing the layer "layer" is disabled. May raise NotFoundError [48] if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

isLayerVisible(...)

isLayerVisible("layer") -> bool>Returns whether the layer "layer" is visible or not, a value of True means that the layer "layer" is visible, a value of False means that the layer "layer" is invisible. May raise NotFoundError [49] if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

sentToLayer(...)

sentToLayer("layer" [, "name"])Sends the object "name" to the layer "layer". The layer must exist. If "name" is not given the currently selected item is used. May raise NotFoundError [50] if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setActiveLayer(...)

setActiveLayer("name")Sets the active layer to the layer named "name". May raise NotFoundError [51] if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setLayerBlendmode(...)

setLayerBlendmode("layer", blend)Sets the layers "layer" blendmode to blend. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setLayerFlow(...)

setLayerFlow("layer", flow)Sets the layers "layer" flowcontrol to flow. If flow is set to true text in layers above this one will flow around objects on this layer. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setLayerLocked(...)

setLayerLocked("layer", locked)Sets the layer "layer" to be locked or not. If locked is set to true the layer will be locked. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setLayerOutlined(...)

setLayerOutlined("layer", outline)Sets the layer "layer" to be locked or not. If outline is set to true the layer will be displayed outlined. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

setLayerPrintable(...)

setLayerPrintable("layer", printable)Sets the layer "layer" to be printable or not. If is the printable set to false the layer won't be printed.

May raise NotFoundError [52] if the layer can't be found. May raise ValueError if the layer

Page related Commands

closeMasterPage(...)

closeMasterPage()Closes the currently active master page, if any, and returns editing to normal. Begin editing with editMasterPage() [47].

createMasterPage(...)

createMasterPage(pageName)Creates a new master page named pageName and opens it for editing.

currentPage(...)

currentPage() -> integerReturns the number of the current working page. Page numbers are counted from 1 upwards, no matter what the displayed first page number of your document is.

deleteMasterPage(...)

deleteMasterPage(pageName)Delete the named master page.

deletePage(...)

deletePage(nr)Deletes the given page. Does nothing if the document contains only one page. Page numbers are counted from 1 upwards, no matter what the displayed first page number is. May raise IndexError if the page number is out of range

editMasterPage(...)

editMasterPage(pageName)Enables master page editing and opens the named master page for editing. Finish editing with closeMasterPage() [48].

getAllObjects(...)

getAllObjects() -> listReturns a list containing the names of all objects on the current page.

getHGuides(...)

getHGuides() -> listReturns a list containing positions of the horizontal guides. Values are in the document's current units - see UNIT_<type> constants.

getPageType(...)

getPageType() -> integerReturns the type of the Page, 0 means left Page, 1 is a middle Page and 2 is a right Page

getVGuides(...)

getVGuides()See getHGuides [49].

getPageItems(...)

getPageItems() -> listReturns a list of tuples with items on the current page. The tuple is: (name, objectType, order) E.g. ['Text1', 4, 0], ['Image1', 2, 1] means that object named 'Text1' is a text frame (type 4) and is the first at the page...

getPageMargins(...)

getPageMargins() Returns the document page margins as a (top, left, right, bottom) tuple in the document's current units. See UNIT_<type> constants and getPageSize [50].

getPageNMargins(...)

getPageNMargins(nr) Returns a tuple with a particular page's margins measured in the document's current units. See UNIT_<type> constants and getPageSize [51].

getPageSize(...)

getPageSize() -> tuple Returns a tuple with document page dimensions measured in the document's current units. See UNIT_<type> constants and getPageMargins [52].

getPageNSize(...)

getPageNSize(nr) -> tuple Returns a tuple with a particular page's size measured in the document's current units. See UNIT_<type> constants and getPageMargins [53].

gotoPage(...)

gotoPage(nr) Moves to the page "nr" (that is, makes the current page "nr"). Note that gotoPage doesn't (currently) change the page the user's view is displaying, it just sets the page that script commands will operate on.
May raise IndexError if the page number is out of range.

importPage(...)

importPage("fromDoc", (pageList, [create, importwhere, importwherePage])) Imports a set of pages (given as a tuple) from an existing document (the file name must be given). This functions maps the "Page->import" dropdown menu function. fromDoc: string; the filename of the document to import pages from pageList: tuple with page numbers of pages to import create: number; 0 to replace existing pages, 1 (default) to insert new pages importWhere: number; the page number (of the current document) at which import the pages importWherePage: number; used if create==1; 0 to create pages before selected page; 1 to create pages after selected page; 2 (default) to create pages at the end of the document

newPage(...)

newPage(where [, "masterpage"]) Creates a new page. If "where" is -1 the new Page is appended to the document, otherwise the new page is inserted before "where". Page numbers are counted from 1 upwards, no matter what the displayed first page number of your document is. The optional parameter "masterpage" specifies the name of the master page for the new page. May raise IndexError if the page number is out of range

pageCount(...)

pageCount() -> integer Returns the number of pages in the document.

redrawAll(...)

redrawAll() Redraws all pages.

savePageAsEPS(...)

savePageAsEPS("name") Saves the current page as an EPS to the file "name".

Layer related Commands

createLayer(...)

createLayer(layer) Creates a new layer with the name "name".
May raise ValueError if the layer name isn't acceptable.

deleteLayer(...)

deleteLayer("layer") Deletes the layer with the name "layer". Nothing happens if the layer doesn't exist or if it's the only layer in the document. May raise NotFoundError [47] if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

getActiveLayer(...)

getActiveLayer() -> string Returns the name of the current active layer.

getLayers(...)

getLayers() -> list Returns a list with the names of all defined layers.

getLayerBlendmode(...)

getLayerBlendmode("layer") -> int Returns the "layer" layer blendmode. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

isLayerFlow(...)

isLayerFlow("layer") -> bool Returns whether text flows around objects on layer "layer", a value of True means that text flows around, a value of False means that the text does not flow around. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

isLayerLocked(...)

isLayerLocked("layer") -> bool Returns whether the layer "layer" is locked or not, a value of True means that the layer "layer" is editable, a value of False means that the layer "layer" is locked. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

getLayerTransparency(...)

getLayerTransparency("layer") -> float Returns the "layer" layer transparency. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

isLayerOutlined(...)

isLayerOutlined("layer") -> bool Returns whether the layer "layer" is outlined or not, a value of True means that the layer "layer" is outlined, a value of False means that the layer "layer" is normal. May raise NotFoundError if the layer can't be found. May raise ValueError if the layer name isn't acceptable.

May raise `ScribusError` if the save failed.

setHGuides(...)

`setHGuides(list)` Sets horizontal guides. Input parameter must be a list of guide positions measured in the current document units - see `UNIT_<type>` constants.
Example: `setHGuides [54](getHGuides [55]() + [200.0, 210.0])` # add new guides without any lost. `setHGuides [56]([90,250])` # replace current guides entirely

setRedraw(...)

`setRedraw(bool)` Disables page redraw when `bool = False`, otherwise redrawing is enabled. This change will persist even after the script exits, so make sure to call `setRedraw [57](True)` in a finally: clause at the top level of your script.

setVGuides(...)

`setVGuides()` See `setHGuides [58]`.

Font Related Commands

getFontNames(...)

getFontNames() -> listReturns a list with the names of all available fonts.

getXFontNames(...)

getXFontNames() -> list of tuplesReturns a larger font info. It's a list of the tuples with: [(Scribus name, Family, Real name, subset (1|0), embed PS (1|0), font file), (...), ...]

renderFont(...)

renderFont("name", "filename", "sample", size, format="PPM") -> boolCreates an image preview of font "name" with given text "sample" and size. If "filename" is not "", image is saved into "filename". Otherwise image data is returned as a string. The optional "format" argument specifies the image format to generate, and supports any format allowed by QPixmap.save(). Common formats are PPM, JPEG, PNG and XPM. May raise NotFoundError [47] if the specified font can't be found. May raise ValueError if an empty sample or filename is passed.

transparent. deleteColor works on the default document colors if there is no document open. In that case, "replace", if specified, has no effect. May raise NotFoundError [50] if a named color wasn't found. May raise ValueError if an invalid color name is specified.

getColor(...)

getColor("name") -> tupleReturns a tuple (C, M, Y, K) containing the four color components of the color "name" from the current document. If no document is open, returns the value of the named color from the default document colors. May raise NotFoundError [51] if the named color wasn't found. May raise ValueError if an invalid color name is specified.

getColorAsRGB(...)

getColorAsRGB("name") -> tupleReturns a tuple (R,G,B) containing the three color components of the color "name" from the current document, converted to the RGB color space. If no document is open, returns the value of the named color from the default document colors. May raise NotFoundError if the named color wasn't found. May raise ValueError if an invalid color name is specified.

getColorNames(...)

getColorNames() -> listReturns a list containing the names of all defined colors in the document. If no document is open, returns a list of the default document colors.

isSpotColor(...)

isSpotColor("name") -> boolReturns True if the color "name" is a spot color. See also setColor().May raise NotFoundError if a named color wasn't found. May raise ValueError if an invalid color name is specified.

replaceColor(...)

replaceColor("name", "replace")Every occurrence of the color "name" is replaced by the color "replace". May raise NotFoundError [52] if a named color wasn't found. May raise ValueError if an invalid color name is specified.

setSpotColor(...)

setSpotColor("name", spot)Set the color "name" as a spot color if spot parameter is True. See also isSpotColor(). May raise NotFoundError if a named color wasn't found. May raise ValueError if an invalid color name is specified.

Creating and Destroying Objects

createCharStyle(...)

createCharStyle(...)Creates a character style. This function takes the following keyword parameters:

- "name" [required] -> name of the char style to create
- "font" [optional] -> name of the font to use
- fontsize [optional] -> font size to set (double)
- "features" [optional] -> nearer typographic details can be defined by a string that might contain the following phrases comma-separated (without spaces): inherit
 - bold
 - italic
 - underline
 - underlinewords
 - strike
 - superscript
 - subscript
 - outline
 - shadowed
 - allcaps
 - smallcaps
- "fillcolor" [optional], "fillshade" [optional] -> specify fill options
- "strokecolor" [optional], "strokeshade" [optional]
- specify stroke options
 - baselineoffset [optional] -> offset of the baseline- shadowxoffset [optional], shadowyoffset [optional] -> offset of the shadow if used
 - outlinewidth [optional] -> width of the outline if used
 - underlineoffset [optional], underlinewidth [optional] -> underline options if used
 - strikethruoffset [optional], strikethruwidth [optional] -> strikethru options if used
 - scaleh [optional], scalev [optional] -> scale of the chars
 - tracking [optional] -> tracking of the text
 - "language" [optional] -> language code

createBezierLine(...)

createBezierLine(list, ["name"]) -> stringCreates a new bezier curve and returns its name. The points for the bezier curve are stored in the list "list" in the following order: [x1, y1, kx1, ky1, x2, y2, kx2, ky2...xn, yn, kxn, kyn] In the points list, x and y mean the x and y coordinates of the point and kx and ky meaning the control point for the curve. The coordinates are given in

the current measurement units of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further access to that object. If "name" is not given Scribus will create one for you.
May raise NameExistsError [47] if you explicitly pass a name that's already used. May raise ValueError if an insufficient number of points is passed or if the number of values passed don't group into points without leftovers.

createEllipse(...)

createEllipse(x, y, width, height, ["name"]) -> stringCreates a new ellipse on the current page and returns its name. The coordinates are given in the current measurement units of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further referencing of that object. If "name" is not given Scribus will create one for you. May raise NameExistsError [48] if you explicitly pass a name that's already used.

createImage(...)

createImage(x, y, width, height, ["name"]) -> stringCreates a new picture frame on the current page and returns its name. The coordinates are given in the current measurement units of the document. "name" should be a unique identifier for the object because you need this name for further access to that object. If "name" is not given Scribus will create one for you. May raise NameExistsError [49] if you explicitly pass a name that's already used.

createLine(...)

createLine(x1, y1, x2, y2, ["name"]) -> stringCreates a new line from the point(x1, y1) to the point(x2, y2) and returns its name. The coordinates are given in the current measurement unit of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further access to that object. If "name" is not given Scribus will create one for you. May raise NameExistsError [50] if you explicitly pass a name that's already used.

createParagraphStyle(...)

createParagraphStyle(...)
Creates a paragraph style. This function takes the following keyword parameters:

- "name" [required] -> specifies the name of the paragraphstyle to create
- linespacingmode [optional] -> specifies the linespacing mode; possible modes are: fixed linespacing: 0
 - automatic linespacing: 1
 - baseline grid linespacing: 2
- linespacing [optional] -> specifies the linespacing if using fixed linespacing
- alignment [optional] -> specifies the alignment of the paragraph left: 0
 - center: 1
 - right: 2
 - justify: 3
 - extend: 4
- leftmargin [optional], rightmargin [optional] -> specify the margin

Color related Commands

changeColor(...)

changeColor("name", c, m, y, k)Changes the color "name" to the specified CMYK value. The color value is defined via four components c = Cyan, m = Magenta, y = Yellow and k = Black. Color components should be in the range from 0 to 255.

Note : deprecated, use changeColorCMYK() instead

May raise NotFoundError [47] if the named color wasn't found. May raise ValueError if an invalid color name is specified.

changeColorRGB(...)

changeColorRGB("name", r, g, b)Changes the color "name" to the specified RGB value. The color value is defined via three components r = red, g = green, b = blue. Color components should be in the range from 0 to 255. May raise NotFoundError [48] if the named color wasn't found. May raise ValueError if an invalid color name is specified.

changeColorCMYK(...)

changeColorCMYK("name", c, m, y, k)Changes the color "name" to the specified CMYK value. The color value is defined via four components c = Cyan, m = Magenta, y = Yellow and k = Black. Color components should be in the range from 0 to 255. May raise NotFoundError [49] if the named color wasn't found. May raise ValueError if an invalid color name is specified.

defineColor(...)

defineColor("name", c, m, y, k)Defines a new color "name". The color Value is defined via three components: c = Cyan, m = Magenta, y = Yellow and k = Black. Color components should be in the range from 0 to 255.

Note : deprecated, use defineColorCMYK() instead.

May raise ValueError if an invalid color name is specified.

defineColorRGB(...)

defineColorRGB("name", r, g, b)Defines a new color "name". The color Value is defined via three components: r = red, g = green, b = blue. Color components should be in the range from 0 to 255.

May raise ValueError if an invalid color name is specified.

defineColorCMYK(...)

defineColorCMYK("name", c, m, y, k)Defines a new color "name". The color Value is defined via three components: c = Cyan, m = Magenta, y = Yellow and k = Black. Color components should be in the range from 0 to 255.

May raise ValueError if an invalid color name is specified.

deleteColor(...)

deleteColor("name", "replace")Deletes the color "name". Every occurrence of that color is replaced by the color "replace". If not specified, "replace" defaults to the color "None" -

setTextScalingH(...)
 setTextScalingH(scale, ["name"]) Sets the horizontal character scaling of the object "name" to "scale" in percent. If "name" is not given the currently selected item is used.

setTextScalingV(...)
 setTextScalingV(scale, ["name"]) Sets the vertical character scaling of the object "name" to "scale" in percent. If "name" is not given the currently selected item is used.

setTextColor(...)
 setTextColor("color", ["name"]) Sets the text color of the text frame "name" to the color "color". If there is some text selected only the selected text is changed. If "name" is not given the currently selected item is used.

setTextShade(...)
 setTextShade(shade, ["name"]) Sets the shading of the text color of the object "name" to "shade". If there is some text selected only the selected text is changed. "shade" must be an integer value in the range from 0 (lightest) to 100 (full color intensity). If "name" is not given the currently selected item is used.

setTextStroke(...)
 setTextStroke("color", ["name"]) Set "color" of the text stroke. If "name" is not given the currently selected item is used.

textOverflows(...)
 textOverflows(["name", nolinks]) -> integer Returns the actual number of overflowing characters in text frame "name". If is nolinks set to non zero value it takes only one frame - it doesn't use text frame linking. Without this parameter it search all linking chain. May raise WrongFrameTypeError if the target frame is not an text frame

- gapbefore [optional], gapafter [optional] -> specify the gaps to the heading and following paragraphs

- firstindent [optional] -> the indent of the first line

- hasdropcap [optional] -> specifies if there are caps (1 = yes, 0 = no)

- dropcaplines [optional] -> height (in lines) of the caps if used

- dropcapoffset [optional] -> offset of the caps if used

- "charstyle" [optional] -> char style to use

createPathText(...)
 createPathText(x, y, "textbox", "beziercurve", ["name"]) -> string Creates a new pathText by merging the two objects "textbox" and "beziercurve" and returns its name. The coordinates are given in the current measurement unit of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further access to that object. If "name" is not given Scribus will create one for you. May raise NameExistsError [51] if you explicitly pass a name that's already used. May raise NotFoundError [52] if one or both of the named base object don't exist.

createPolyLine(...)
 createPolyLine(list, ["name"]) -> string Creates a new polyline and returns its name. The points for the polyline are stored in the list "list" in the following order: [x1, y1, x2, y2...xn, yn]. The coordinates are given in the current measurement units of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further access to that object. If "name" is not given Scribus will create one for you. May raise NameExistsError [53] if you explicitly pass a name that's already used. May raise ValueError if an insufficient number of points is passed or if the number of values passed don't group into points without leftovers.

createPolygon(...)
 createPolygon(list, ["name"]) -> string Creates a new polygon and returns its name. The points for the polygon are stored in the list "list" in the following order: [x1, y1, x2, y2...xn, yn]. At least three points are required. There is no need to repeat the first point to close the polygon. The polygon is automatically closed by connecting the first and the last point. The coordinates are given in the current measurement units of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further access to that object. If "name" is not given Scribus will create one for you. May raise NameExistsError [54] if you explicitly pass a name that's already used. May raise ValueError if an insufficient number of points is passed or if the number of values passed don't group into points without leftovers.

createRect(...)
 createRect(x, y, width, height, ["name"]) -> string Creates a new rectangle on the current page and returns its name. The coordinates are given in the current measurement units of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name to reference that object in future. If "name" is not given Scribus will create one for you. May raise NameExistsError [55] if you explicitly pass a name that's already used.

createText(...)

`createText(x, y, width, height, ["name"])` -> stringCreates a new text frame on the actual page and returns its name. The coordinates are given in the actual measurement unit of the document (see UNIT constants). "name" should be a unique identifier for the object because you need this name for further referencing of that object. If "name" is not given Scribus will create one for you. May raise `NameExistsError` [56] if you explicitly pass a name that's already used.

deleteObject(...)

`deleteObject(["name"])`Deletes the item with the name "name". If "name" is not given the currently selected item is deleted.

getAllStyles(...)

`getAllStyles()` -> listReturn a list of the names of all paragraph styles in the current document.

objectExists(...)

`objectExists(["name"])` -> boolTest if an object with specified name really exists in the document. The optional parameter is the object name. When no object name is given, returns True if there is something selected.

setColumnGap(...)

`setColumnGap(size, ["name"])`Sets the column gap of the text frame "name" to the value "size". If "name" is not given the currently selected item is used.
May throw `ValueError` if the column gap is out of bounds (must be positive).

setFont(...)

`setFont("font", ["name"])`Sets the font of the text frame "name" to "font". If there is some text selected only the selected text is changed. If "name" is not given the currently selected item is used. May throw `ValueError` if the font cannot be found.

setFontSize(...)

`setFontSize(size, ["name"])`Sets the font size of the text frame "name" to "size". "size" is treated as a value in points. If there is some text selected only the selected text is changed. "size" must be in the range 1 to 512. If "name" is not given the currently selected item is used. May throw `ValueError` for a font size that's out of bounds.

setLineSpacing(...)

`setLineSpacing(size, ["name"])`Sets the line spacing ("leading") of the text frame "name" to "size". "size" is a value in points. If "name" is not given the currently selected item is used. May throw `ValueError` if the line spacing is out of bounds.

setLineSpacingMode(...)

`setLineSpacingMode(mode, ["name"])`Sets the line spacing mode of the text frame "name" to "mode". If "name" is not given the currently selected item is used. Mode values are the same as in `createParagraphStyle`. May throw `ValueError` if the line spacing mode is out of bounds.

setPDFBookmark(...)

`setPDFBookmark("toggle", ["name"])`Sets whether (toggle = 1) the text frame "name" is a bookmark or not. If "name" is not given the currently selected item is used. May raise `WrongFrameTypeError` if the target frame is not a text frame

setText(...)

`setText("text", ["name"])`Sets the text of the text frame "name" to the text of the string "text". Text must be UTF8 encoded - use e.g. `unicode(text, 'iso-8859-2')`. See the FAQ for more details. If "name" is not given the currently selected item is used.

setTextAlignment(...)

`setTextAlignment(align, ["name"])`Sets the text alignment of the text frame "name" to the specified alignment. If "name" is not given the currently selected item is used. "align" should be one of the `ALIGN_` constants defined in this module - see `dir(scribus)`. May throw `ValueError` for an invalid alignment constant.

setTextDistances(...)

`setTextDistances(left, right, top, bottom, ["name"])`Sets the text distances of the text frame "name" to the values "left", "right", "top" and "bottom". If "name" is not given the currently selected item is used. May throw `ValueError` if any of the distances are out of bounds (must be positive).

getLineColor(["name"]) -> string>Returns the name of the line color of the object "name". If "name" is not given the currently selected item is used.

getTextDistances(...)

getTextDistances(["name"]) -> tuple>Returns the text distances of the text frame "name" expressed in points. The distances are returned as a tuple like (left, right, top, bottom). If "name" is not given the currently selected item is used.

getTextLength(...)

getTextLength(["name"]) -> integer>Returns the length of the text in the text frame "name". If "name" is not given the currently selected item is used.

getTextLines(...)

getTextLines(["name"]) -> integer>Returns the number of lines of the text in the text frame "name". If "name" is not given the currently selected item is used.

getTextShade(...)

getTextShade(["name"]) -> integer>Returns the shading value of the line color of the object "name". If "name" is not given the currently selected item is used.

hyphenateText(...)

hyphenateText(["name"]) -> boolDoes hyphenation on text frame "name". If "name" is not given the currently selected item is used.
May raise `WrongFrameTypeError` if the target frame is not a text frame

insertText(...)

insertText("text", pos, ["name"]) Inserts the text "text" at the position "pos" into the text frame "name". Text must be UTF encoded (see `setText[47]()` as reference) The first character has an index of 0. Inserting text at position -1 appends it to the frame. If "name" is not given the currently selected item is used.
May throw `IndexError` for an insertion out of bounds.

isPDFBookmark(...)

isPDFBookmark(["name"]) -> bool>Returns true if the text frame "name" is a PDF bookmark. If "name" is not given the currently selected item is used.
May raise `WrongFrameTypeError` if the target frame is not a text frame

selectText(...)

selectText(start, count, ["name"]) Selects "count" characters of text in the text frame "name" starting from the character "start". Character counting starts at 0. If "count" is zero, any text selection will be cleared. If "name" is not given the currently selected item is used.
May throw `IndexError` if the selection is outside the bounds of the text.

setColumns(...)

setColumns(nr, ["name"]) Sets the number of columns of the text frame "name" to the integer "nr". If "name" is not given the currently selected item is used.
May throw `ValueError` if number of columns is not at least one.

Selecting Objects

deselectAll(...)

deselectAll() Deselects all objects in the whole document.

getSelectedObject(...)

getSelectedObject(nr) -> string>Returns the name of the selected object. "nr" if given indicates the number of the selected object, e.g. 0 means the first selected object, 1 means the second selected Object and so on.

moveSelectionToBack(...)

moveSelectionToBack() Moves the current selection to the back.

moveSelectionToFront(...)

moveSelectionToFront() Moves the current selection to the front.

selectionCount(...)

selectionCount() -> integer>Returns the number of selected objects.

selectObject(...)

selectObject("name") Selects the object with the given "name".

Handling Text Frames

dehyphenateText(...)

dehyphenateText(["name"]) -> bool Does dehyphenation on text frame "name". If "name" is not given the currently selected item is used.
May raise WrongFrameTypeError if the target frame is not a text frame

deleteText(...)

deleteText(["name"]) Deletes any text in the text frame "name". If there is some text selected, only the selected text will be deleted. If "name" is not given the currently selected item is used.

getAllText(...)

getAllText(["name"]) -> string Returns the text of the text frame "name" and of all text frames which are linked with this frame. If this text frame has some text selected, the selected text is returned. If "name" is not given the currently selected item is used.

getColumnGap(...)

getColumnGap(["name"]) -> float Returns the column gap size of the text frame "name" expressed in points. If "name" is not given the currently selected item is used.

getColumns(...)

getColumns(["name"]) -> integer Gets the number of columns of the text frame "name". If "name" is not given the currently selected item is used.

getFont(...)

getFont(["name"]) -> string Returns the font name for the text frame "name". If this text frame has some text selected the value assigned to the first character of the selection is returned. If "name" is not given the currently selected item is used.

getFontSize(...)

getFontSize(["name"]) -> float Returns the font size in points for the text frame "name". If this text frame has some text selected the value assigned to the first character of the selection is returned. If "name" is not given the currently selected item is used.

getLineSpacing(...)

getLineSpacing(["name"]) -> float Returns the line spacing ("leading") of the text frame "name" expressed in points. If "name" is not given the currently selected item is used.

getText(...)

getText(["name"]) -> string Returns the text of the text frame "name". If this text frame has some text selected, the selected text is returned. All text in the frame, not just currently visible text, is returned. If "name" is not given the currently selected item is used.

getTextColor(...)

scales the group to 50 % of its original size, a value of 1.5 scales the group to 150 % of its original size. The value for "factor" must be greater than 0. If "name" is not given the currently selected item is used. May raise ValueError if an invalid scale factor is passed.

setScaleImageToFrame(...)

setScaleImageToFrame(scaletoframe, proportional=None, name=<selection>)Sets the scale to frame on the selected or specified image frame to 'scaletoframe'. If 'proportional' is specified, set fixed aspect ratio scaling to 'proportional'. Both 'scaletoframe' and 'proportional' are boolean. May raise WrongFrameTypeError.

setStyle(...)

setStyle("style" [, "name"])Apply the named "style" to the object named "name". If object name is given, style is applied to the current text selection in object "name". If no object name is given, style is applied on selected object.

sizeObject(...)

sizeObject(width, height [, "name"])Resizes the object "name" to the given width and height. If "name" is not given the currently selected item is used.

unGroupObject(...)

unGroupObjects("name")Destructs the group the object "name" belongs to. If "name" is not given the currently selected item is used.

textFlowMode(...)

textFlowMode("name" [, state])Enables/disables "Text Flows Around Frame" feature for object "name". Called with parameters string name and optional int "state" (0 <= state <= 3). Setting "state" to 0 will disable text flow. Setting "state" to 1 will make text flow around object frame. Setting "state" to 2 will make text flow around bounding box. Setting "state" to 3 will make text flow around contour line. If "state" is not passed, text flow is toggled.

Setting Object Properties

linkTextFrames(...)

linkTextFrames("fromname", "toname")Link two text frames. The frame named "fromname" is linked to the frame named "toname". The target frame must be an empty text frame and must not link to or be linked from any other frames already.
May throw ScribusException [47] if linking rules are violated.

loadImage(...)

loadImage("filename" [, "name"])Loads the picture "picture" into the image frame "name". If "name" is not given the currently selected item is used.
May raise WrongFrameTypeError [48] if the target frame is not an image frame

setFillBlendmode(...)

setFillBlendmode(blendmode, ["name"])Sets the fill blendmode of the object "name" to blendmode is the name of one of the defined colors. If "name" is not given the currently selected item is used.

setCornerRadius(...)

setCornerRadius(radius, ["name"])Sets the corner radius of the object "name". The radius is expressed in points. If "name" is not given the currently selected item is used.
May raise ValueError if the corner radius is negative.

setFillColor(...)

setFillColor("color", ["name"])Sets the fill color of the object "name" to the color "color". "color" is the name of one of the defined colors. If "name" is not given the currently selected item is used.

setFillShade(...)

setFillShade(shade, ["name"])Sets the shading of the fill color of the object "name" to "shade". "shade" must be an integer value in the range from 0 (lightest) to 100 (full Color intensity). If "name" is not given the currently selected item is used.
May raise ValueError if the fill shade is out of bounds.

setFillTransparency(...)

setFillTransparency(transparency, ["name"])Sets the fill transparency of the object "name" to transparency is the name of one of the defined colors. If "name" is not given the currently selected item is used.

setLineBlendmode(...)

setLineBlendmode(blendmode, ["name"])Sets the line blendmode of the object "name" to blendmode is the name of one of the defined colors. If "name" is not given the currently

selected item is used.

setLineTransparency(...)

setLineTransparency(transparency, ["name"]) Sets the line transparency of the object "name" to transparency. If "name" is the name of one of the defined colors. If "name" is not given the currently selected item is used.

setGradientFill(...)

setGradientFill(type, "color1", "color2", shade1, "color2", shade2, ["name"]) Sets the gradient fill of the object "name" to type. Color descriptions are the same as for setFillColor [49]() and setFillShade [50](). See the constants for available types (FILL_<type>).

setGradientStop(...)

setGradientStop("color", shade, opacity, rampoint, ["name"]) Set or add a gradient stop to the gradient fill of the object "name" at position rampoint. Color descriptions are the same as for setFillColor [51]() and setFillShade [52](). setGradientFill() must have been called previously for the gradient fill to be visible.

setLineCap(...)

setLineCap(captype, ["name"]) Sets the line cap style of the object "name" to the style "cap". If "name" is not given the currently selected item is used. There are predefined constants for "cap" - CAP_<type>.

setLineColor(...)

setLineColor("color", ["name"]) Sets the line color of the object "name" to the color "color". If "name" is not given the currently selected item is used.

setMultiLine(...)

setMultiLine("namedStyle", ["name"]) Sets the line style of the object "name" to the named style "namedStyle". If "name" is not given the currently selected item is used. May raise NotFoundError [53] if the line style doesn't exist.

setLineJoin(...)

setLineJoin(join, ["name"]) Sets the line join style of the object "name" to the style "join". If "name" is not given the currently selected item is used. There are predefined constants for join - JOIN_<type>.

setLineShade(...)

setLineShade(shade, ["name"]) Sets the shading of the line color of the object "name" to "shade". "shade" must be an integer value in the range from 0 (lightest) to 100 (full color intensity). If "name" is not given the currently selected item is used. May raise ValueError if the line shade is out of bounds.

setLineStyle(...)

setLineStyle(style, ["name"]) Sets the line style of the object "name" to the style "style". If "name" is not given the currently selected item is used. There are predefined constants for "style" - LINE_<style>.

setLineWidth(...)

Manipulating Objects

duplicateObject(...)

duplicateObject(["name"]) -> string Creates a Duplicate of the selected Object (or Selection Group).

groupObjects(...)

groupObjects(list) -> string Groups the objects named in "list" together. "list" must contain the names of the objects to be grouped. If "list" is not given the currently selected items are used. Returns the group name for further referencing.

isLocked(...)

isLocked(["name"]) -> bool Returns true if the object "name" is locked. If "name" is not given the currently selected item is used.

lockObject(...)

lockObject(["name"]) -> bool Locks the object "name" if it's unlocked or unlock it if it's locked. If "name" is not given the currently selected item is used. Returns true if locked.

moveObject(...)

moveObject(dx, dy [, "name"]) Moves the object "name" by dx and dy relative to its current position. The distances are expressed in the current measurement unit of the document (see UNIT constants). If "name" is not given the currently selected item is used. If the object "name" belongs to a group, the whole group is moved.

moveObjectAbs(...)

moveObjectAbs(x, y [, "name"]) Moves the object "name" to a new location. The coordinates are expressed in the current measurement unit of the document (see UNIT constants). If "name" is not given the currently selected item is used. If the object "name" belongs to a group, the whole group is moved.

rotateObject(...)

rotateObject(rot [, "name"]) Rotates the object "name" by "rot" degrees relatively. The object is rotated by the vertex that is currently selected as the rotation point - by default, the top left vertex at zero rotation. Positive values mean counter clockwise rotation when the default rotation point is used. If "name" is not given the currently selected item is used.

rotateObjectAbs(...)

rotateObjectAbs(rot [, "name"]) Sets the rotation of the object "name" to "rot". Positive values mean counter clockwise rotation. If "name" is not given the currently selected item is used.

scaleGroup(...)

scaleGroup(factor [, "name"]) Scales the group the object "name" belongs to. Values greater than 1 enlarge the group, values smaller than 1 make the group smaller e.g a value of 0.5

"name" is not given the currently selected item is used.

getLineJoin(...)

getLineJoin(["name"]) -> integer (see constants)Returns the line join style of the object "name". If "name" is not given the currently selected item is used. The join types are: JOIN_BEVEL, JOIN_MITTER, JOIN_ROUND

getLineShade(...)

getLineShade(["name"]) -> integerReturns the shading value of the line color of the object "name". If "name" is not given the currently selected item is used.

getLineStyle(...)

getLineStyle(["name"]) -> integer (see constants)Returns the line style of the object "name". If "name" is not given the currently selected item is used. Line style constants are: LINE_DASH, LINE_DASHDOT, LINE_DASHDOTDOT, LINE_DOT, LINE_SOLID

getLineTransparency(...)

getLineTransparency(["name"]) -> floatReturns the line transparency of the object "name". If "name" is not given the currently selected item is used.

getLineWidth(...)

getLineWidth(["name"]) -> integerReturns the line width of the object "name". If "name" is not given the currently selected item is used.

getPosition(...)

getPosition(["name"]) -> (x,y)Returns a (x, y) tuple with the position of the object "name". If "name" is not given the currently selected item is used. The position is expressed in the actual measurement unit of the document - see UNIT_<type> for reference.

getRotation(...)

getRotation(["name"]) -> integerReturns the rotation of the object "name". The value is expressed in degrees, and clockwise is positive. If "name" is not given the currently selected item is used.

getSize(...)

getSize(["name"]) -> (width,height)Returns a (width, height) tuple with the size of the object "name". If "name" is not given the currently selected item is used. The size is expressed in the current measurement unit of the document - see UNIT_<type> for reference.

setLineWidth(width, ["name"])Sets line width of the object "name" to "width". "width" must be in the range from 0.0 to 12.0 inclusive, and is measured in points. If "name" is not given the currently selected item is used.

May raise ValueError if the line width is out of bounds.

scaleImage(...)

scaleImage(x, y [, "name"])Sets the internal scaling factors of the picture in the image frame "name". If "name" is not given the currently selected item is used. A number of 1 means 100 %. Internal scaling factors are different from the values shown on properties palette. Note : deprecated, use setImageScale() instead.

May raise WrongFrameTypeError [54] if the target frame is not an image frame

setImageScale(...)

setImageScale(x, y [, "name"])Sets the scaling factors of the picture in the image frame "name". If "name" is not given the currently selected item is used. A number of 1 means 100 %. Scaling factors are equal to the values shown on properties palette.

May raise WrongFrameTypeError [55] if the target frame is not an image frame

setImageOffset(...)

setImageOffset(x, y [, "name"])Sets the position of the picture in the image frame "name". If "name" is not given the currently selected item is used. The specified offset values are equal to the values shown on properties palette when point unit is used.

May raise WrongFrameTypeError [56] if the target frame is not an image frame

traceText(...)

traceText(["name"])Convert the text frame "name" to outlines. If "name" is not given the currently selected item is used.

unlinkTextFrames(...)

unlinkTextFrames("name")Remove the specified (named) object from the text frame flow/linkage. If the frame was in the middle of a chain, the previous and next frames will be connected. eg 'a->b->c' becomes 'a->c' when you unlinkTextFrames [57](b)
May throw ScribusException [58] if linking rules are violated.

Getting Object Properties

getObjectType(...)

getObjectType(["name"]) -> string Get type of object "name" as a string. If "name" is not given the currently selected item is used.

getCornerRadius(...)

getCornerRadius(["name"]) -> integer Returns the corner radius of the object "name". The radius is expressed in points. If "name" is not given the currently selected item is used.

getFillColor(...)

getFillColor(["name"]) -> string Returns the name of the fill color of the object "name". If "name" is not given the currently selected item is used.

getFillBlendmode(...)

getFillBlendmode(["name"]) -> integer Returns the fill blendmode of the object "name". If "name" is not given the currently selected item is used.

getFillShade(...)

getFillShade(["name"]) -> integer Returns the shading value of the fill color of the object "name". If "name" is not given the currently selected item is used.

getFillTransparency(...)

getFillTransparency(["name"]) -> float Returns the fill transparency of the object "name". If "name" is not given the currently selected item is used.

getImageFile(...)

getImageName(["name"]) -> string Returns the filename for the image in the image frame. If "name" is not given the currently selected item is used.

getImageScale(...)

getImageScale(["name"]) -> (x,y) Returns a (x, y) tuple containing the scaling values of the image frame "name". If "name" is not given the currently selected item is used.

getLineBlendmode(...)

getLineBlendmode(["name"]) -> integer Returns the line blendmode of the object "name". If "name" is not given the currently selected item is used.

getLineCap(...)

getLineEnd(["name"]) -> integer (see constants) Returns the line cap style of the object "name". If "name" is not given the currently selected item is used. The cap types are: CAP_FLAT, CAP_ROUND, CAP_SQUARE

getLineColor(...)

getLineColor(["name"]) -> string Returns the name of the line color of the object "name". If